Lektion 2

Kap.2 Mathematische Grundlagen

Inhalt

- Reihen
- Potenzen
- Logarithmen
- sonstige Notationen (Runden, Intervalle)
- Graphen und Bäume

2.1 Reihen

Bei der Analyse der Komplexität von Algorithmen brauchen wird öfter mal die Bildung von Summen von bestimmten Zahlenfolgen, in der Mathematik *Reihen* genannt.

Arithmetische Reihe

- **Allgemeine arithmetische Reihe**: $a_0+(a_0+d)+(a_0+2d)+ \ldots +(a_0+n\cdot d)$

  $$\sum_{i=0}^{n}(a_0+ i\cdot d)=(n+1)(a_0+ d\frac{n}{2})$$

  Beispiel: Summe der ungeraden Zahlen von 1 bis 99, d.h.1 + 3 + 5 + … + 99:
  $a_0 = 1$
  d = 2
  n = 49
  Ergebnis: 50 * (1 + 2*49/2) = 2500

- **Gaußsche Summenformel**: 1 + 2 + 3 + ... + *n*, also Summe der natürlichen Zahlen von 1 bis *n*. Dies ist der Spezialfall mit $a_0 = 0$; d = 1.

  $$\sum_{i=1}^{n} i=\frac{n(n+1)}{2}$$

  Beispiel: Summe der Zahlen von 1 bis 50:
  50 * 51 / 2 = 1275

wichtig

Die allgemeine arithmetische Reihe werden wir eher selten benötigen, der Gaußsche Spezialfall taucht aber relativ häufig auf, z.B. zur Berechnung der Schleifendurchläufe bei geschachtelten Schleifen, wenn die äußere Schleife mit Laufvariable *i* von 1 bis *n* läuft und die innere Schleife dann jeweils von 1 bis *i* geht.

Geometrische Reihe

- **Summe von Potenzen.** allgemeine Form: $k^0 + k^1 + k^2 + ... + k^n$

  $$\sum_{i=0}^{n} k^i = \frac{k^{n+1}-1}{k-1}$$

  Für Algorithmen&Datenstrukturen ist vor allem der Spezialfall der Zweierpotenzen wichtig.

- Spezialfall **Summe von Zweierpotenzen**: $1 + 2 + 4 + 8 + ... + 2^n$

  $$\sum_{i=0}^{n} 2^i = 2^{n+1}-1$$

  Die Summe der Zweiterpotenzen bis 2^n ergibt 2^{n+1} - 1, d.h. die nächste Zweierpotenz minus 1 (das sollte jeder Informatiker im Kopf haben).

  Beispiel:

  $1 + 2 + 4 + 8 + ... + 512$
  $= 2^0 + 2^1 + ... + \mathbf{2^9}$
  $= \mathbf{2^{10}} - 1$
  $= 1023$

Aufgabe 2.1 - 2er-Potenzen

a) Die Werte der 2er-Potenzen von 2^0 bis 2^{10} sollten Sie jederzeit auswendig wissen. Wie lauten sie?

b) Was ist grob abgeschätzt 2^{20}, 2^{30}, 2^{16} und 2^{32}?

2.2 Potenzen und Logarithmen

Eigenschaften von Potenzen

Die grundlegenden Rechengesetze kennen Sie noch aus der Schule:

- $a^x \cdot a^y = a^{(x+y)}$
- $\frac{a^x}{a^y} = a^{(x-y)}$

- $(a^x)^y = a^{x \cdot y}$

Eigenschaften von Logarithmen

Definition: Logarithmus ist die Umkehrung der Potenzierung

$$y = \log_b(x) \;\Leftrightarrow\; b^y = x$$

Logarithmus-Gesetze:

- $\log_b(x \cdot y) = \log_b x + \log_b y$
- $\log_b(\frac{x}{y}) = \log_b x - \log_b y$
- $\log_b(x^y) = y \cdot \log_b x$

Notation für spezielle Logarithmen

Folgende Bezeichnungen werden wir auch verwenden:

ld *x*	= $\log_2(x)$	Logarithmus zur Basis 2 (logarithmus dualis)
ln *x*	= $\log_e(x)$	Logarithmus zur Basis e (logarithmus naturalis)
lg *x*	= $\log_{10}(x)$	Logarithmus zur Basis 10 (dekadischer Logarithmus)

Basiswechsel bei Logarithmen

Logarithmen mit unterschiedlichen Basen lassen sich folgendermaßen ineinander umrechnen:

$$\log_b(x) = \frac{1}{\log_a(b)} \cdot \log_a(x)$$

D.h. Werte von Logarithmen $\log_a(x)$ und $\log_b(x)$ zu verschiedenen Basen *a* und *b* lassen sich einfach durch einen konstanten Faktor ineinander umrechnen, der unabhängig vom Argument *x* ist.

Merkregel: Verhältnis zwischen Logarithmen zur Basis 2 und Basis 10 ist ca. 3,3.

ld x ≈ **3,3**·lg x

Aufgabe 2.2 - Zweierlogarithmus abschätzen

- Schätzen Sie ab (ohne Rechner):

  ld 1 000
  ld 1 000 000
  ld 1 000 000 000

2.3 Notationskonventionen

Auf- und Abrunden

⌈x⌉ zur nächsten ganzen Zahl **aufrunden**
⌊x⌋ zur nächsten ganzen Zahl **abrunden**

Intervallangaben

[a .. b] = {x | a ≤ x ∧ x ≤ b} mit Intervallgrenzen
]a .. b[= {x | a < x ∧ x < b} ohne Intervallgrenzen

Teilbereiche von Arrays

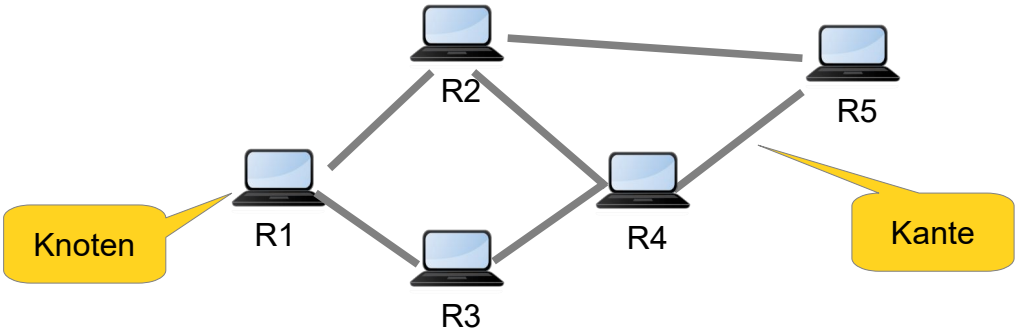
arr[i .. k] Teilfolge der Elemente von arr[i] bis arr[k]

2.4 Grundbegriffe der Graphentheorie

Graphen sind mathematische Strukturen, die aus einer Menge von Knoten und einer Menge von Kanten bestehen.

Durch einen Graphen wird beschrieben, welche Zweierbeziehungen (repräsentiert als sog. **Kanten**, engl. **edges**) zwischen Elementen (bezeichnet als **Knoten**, engl. **vertices** oder **nodes**) bestehen. Je nachdem, ob Kanten eine gerichtete oder ungerichtete Verbindung zwischen Knoten repräsentieren, spricht man von gerichtete bzw. ungerichteten Graphen.

Beispiel: Repräsentation eines Rechnernetzes als (ungerichteter) Graph:



2.4.1 Gerichtete und ungerichtete Graphen

Mathematisch wird ein **Graph G = (V, E)** als eine Menge V (engl. *vertices*) von **Knoten** und eine Menge E von **Kanten** (engl. *edges*) definiert. Man unterscheidet zwischen **gerichteten** und **ungerichteten** Graphen.

- Bei einem **gerichteten Graphen** ist jede Kante ein Paar (s,t) aus zwei Knoten, dem Startknoten s und dem Zielknoten t.

  gerichteter Graph G = (V, E)
  V Menge von Knoten
  E Menge von Kanten
  Jede Kante e $\in$ E ist ein **Paar** (s,t) $\in$ V x V

- Bei **ungerichteten Graphen** wird eine Kante als eine Menge {s, t} aus Start- und Zielknoten formal beschrieben. Im Gegensatz zu Paaren haben Menge keine Reihenfolge, d.h. die Menge {s,t} ist gleich der Menge {t,s}. Somit kann nicht zwischen Start- und Zielknoten einer Kante unterschieden werden.

  ungerichteter Graph G = (V, E)
  V Menge von Knoten
  E Menge von Kanten,
  Jede Kante e $\in$ E ist ein **Menge** $\{v_1, v_2\} \subseteq V$

Beispiel: Das Rechnernetz oben ist ein ungerichteter Graph G = (V, E) mit
V = {R1, R2, R3, R4, R5} Knoten
E = { {R1,R2}, {R1,R3}, {R2,R4}, {R2,R5}, {R4, R5} } Kanten

- Kanten können bei sog. *gewichteten Graphen* auch mit einer Zahl als "Gewicht" versehen sein.

2.4.2 Diagrammdarstellung von Graphen

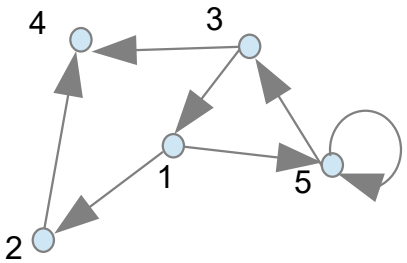
Oft werden Graphen nicht formal als Menge von Knoten und Kanten dargestellt, sondern in Diagrammform gezeichnet.

In der Diagrammdarstellung (wie beim Rechnernetz-Beispiel oben) werden die Kanten als Linien dargestellt – bei gerichteten Graphen als Pfeil vom Start zum Zielknoten.

Beispiel 2.3 - Darstellung gerichteter und ungerichtete Graphen

- **Gerichtete Graph**

  G = (V, E) mit
  V = {1,2,3,4,5}
  E = { (1,2), (1,5), (2,4), (3,1), (3,4), (5,3), (5,5) }



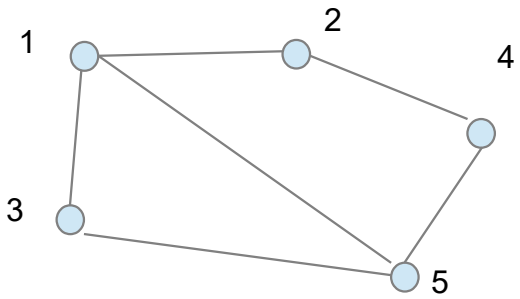
Kanten, die einen Knoten mit sich selbst verbinden, sind erlaubt und werden als **Schlingen** bezeichnet.

- **Ungerichteter Graph**

  G = (V, E) mit

  V = {1,2,3,4,5},

  E = { {1,2}, {1,3} { 1,5}, {2,4}, {3,5}, {4,5} }



Anmerkung

- Graphen im Sinn der Graphentheorie haben nichts mit dem Begriff des *Funktionsgraphen* (Schaubild einer Funktion) zu tun, den Sie aus den Mathematik-Vorlesungen kennen.

Graphen und Graph-Layout (Diagrammdarstellung)

Ein Graph kann auf unterschiedliche Weise zeichnerisch in der Ebene dargestellt werden (Diagrammdarstellung in der Ebene, Graph-Layout):

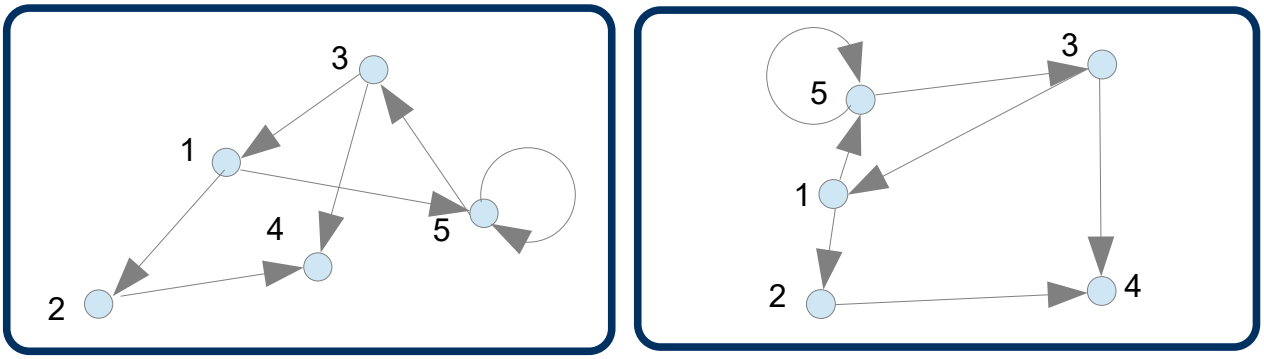
Beispiel 2.4 - Graph-Layout

- Sei Graph G = (V, E) gegeben, wobei

  V = {1,2,3,4,5}

  E = {(1,2), (1,5), (2,4), (3,4), (3,1), (5,3), (5,5) }

Zwei Diagrammdarstellungen (Graph-Layouts) von G:



2.4.3 Einige Grundbegriffe für Graphen

Mit dem Grad eines Knotens wird bezeichnet, wie viele Kanten an diesem Knoten enden. Bei gerichteten Graphen kann man außerdem noch Eingangs- und Ausgangsgrad unterscheiden, d.h. wie viele Kanten hineinführen und wie viele heraus.

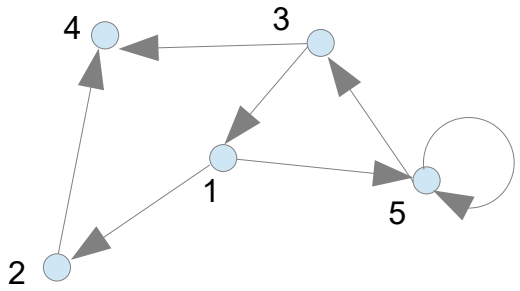
Definition 2.5 - Grad eines Knotens

Für einen Knoten v eines gerichteten Graphen $G = (V, E)$ ist

- der **Eingangsgrad** die Anzahl der Kanten mit Zielknoten v
- der **Ausgangsgrad** die Anzahl der Kanten mit Ausgangsknoten v
- der **Grad** die Summe aus Eingangs- und Ausgangsgrad von v

Aufgabe 2.6 - Grad von Knoten

Welchen Eingangsgrad, Ausgangsgrad und Grad haben die Knoten 1, 4 und 5?



Stellt man sich einen Graph als eine Art Landkarte vor, so dass die Kanten die Straßen sind, die man entlang gehen kann (bei gerichteten Graphen nur in eine Richtung, bei ungerichteten Graphen in beide Richtungen), ergibt sich naheliegend der Begriff des *Wegs* (oder *Pfads*): Ein Weg/Pfad wird durch einfach ein Folge von Knoten beschrieben, so dass man jeweils über eine Kante von einem Knoten zum nächsten kommt. Die *Länge* eines Pfads ist die Anzahl der Kanten, die dabei durchlaufen werden.

Definition 2.7 - Pfad/Weg, Zyklus

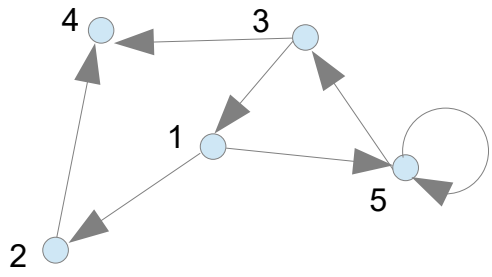
gegeben: gerichteter oder ungerichteter Graph $G = (V, E)$.

- Eine Folge von Knoten $(v_0, v_1, ..., v_n)$ mit Kanten von v_i nach v_{i+1} für $0 \leq i < n$, $n \geq 0$ heißt **Weg** / **Pfad** (engl. *path*) der **Länge** n von v_0 nach v_n.
- Ein **Zyklus** ist ein Weg $(v_0, v_1, ..., v_n)$ mit $n > 0$ und $v_0 = v_n$, d.h. der Weg endet wieder mit dem Startknoten.
- Ein gerichteter Graph heißt **azyklischer Graph** (engl. *directed acyclic graph*, *DAG*), wenn der keinen Zyklus enthält.

Pfade, die wieder am Ausgangsknoten enden, werden als *Zyklen* bezeichnet. Graphen, die keinen Zyklus enthalten werden als *azyklisch* bezeichnet.

Aufgabe 2.8 - Grundbegriffe der Graphentheorie

Gegeben ist folgender gerichtete Graph *G*:



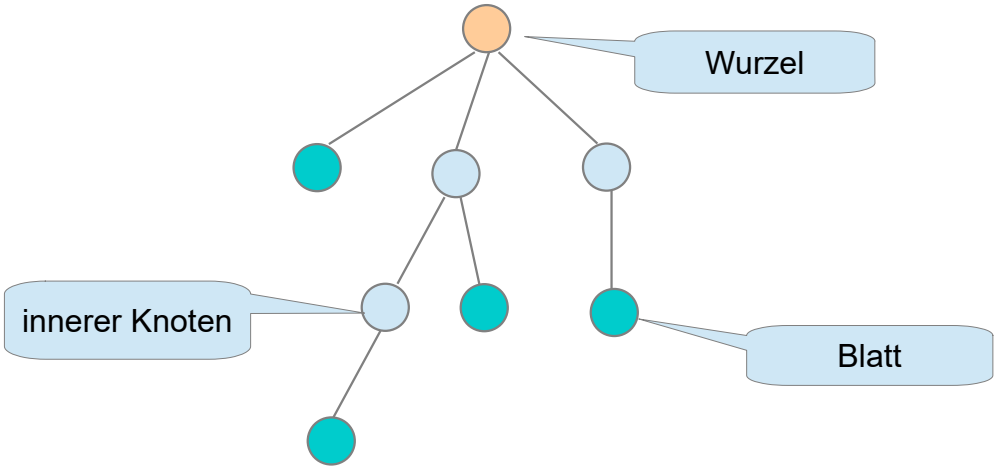
a) Geben Sie dazu einen **Pfad** der Länge 3 an.

b) Gibt es einen **Zyklus** der Länge 3?

c) Ist der Graph *G* **azyklisch**?

2.4.4 Bäume

Um streng hierarchisch aufgebaut Strukturen zu beschreiben, wird in der Informatik oft eine spezielle Art von Graphen verwendet, die **Bäume**.



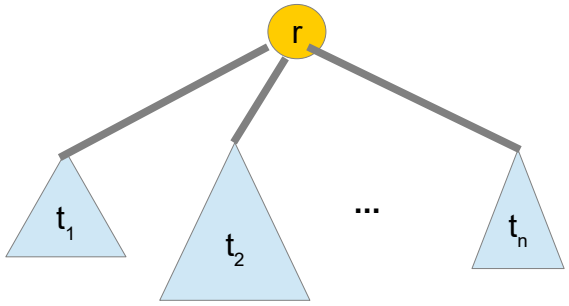
- Ein (gewurzelter) Baum hat einen Knoten als **Wurzel**.
- Jeder Knoten kann beliebig viele Nachfolger haben. Die Nachfolger eines Knotens nennt man seine **Kinder** (bzw. Söhne/Töchter).
- Jeder Knoten, außer der Wurzel, hat genau einen Knoten als **Elternknoten** (Vorgänger)
- Knoten ohne Kinder heißen **Blätter.**

- Knoten, die keine Blätter sind, heißen **innere Knoten.** (Besteht der Baum aus mehr als nur einem Knoten, zählt auch die Wurzel als innerer Knoten.)
- Bäume werden üblicherweise so dargestellt, dass die Wurzel oben und die Blätter unten sind.

Bäume als rekursive Datenstruktur

Bäume können als eine rekursiv definierte Datenstruktur gesehen werden, entsprechend folgender (rekursiver) Definition:

- Der leere Graph (mit leerer Knoten- und Kantenmenge) ist ein Baum
- Sind t_1 , t_2, … t_n Bäume und ist r ein neuer Knoten, dann ist folgendes auch ein Baum (mit Wurzel r):



2.4.5 Eigenschaften von Bäumen

Bei der Analyse von baumartigen Datenstrukturen wird die Höhe von Bäumen eine wichtige Rolle spielen. Deshalb finden Sie hier nochmals die entsprechende Definition und einige damit zusammenhängende Eigenschaften.

Definition 2.9 - Höhe eines Baums

- Ein leerer Baum t hat die Höhe
  $h(t) = 0$
- Ein nicht leerer Baum t mit einer Wurzel und mit den Teilbäumen t_1 bis t_n hat die Höhe
  $h(t) = 1 + \max\{ h(t_1), \ldots, h(t_n) \}$.

Anmerkung

In der Literatur finden sich zwei leicht voneinander abweichende Definitionen für die Höhe von Bäumen.

a) In der oben angegebenen Definition, die wir **immer in diesem Semester verwenden**, hat ein ganz leerer Baum (mit leerer Knoten- und Kantenmenge) die Höhe 0 und ein Baum, der nur aus der Wurzel besteht, die Höhe 1.

Die Höhe entspricht dann der Anzahl der *Knoten* des längsten Pfads von der Wurzel zu einem Blatt.

b) Eine andere auch gebräuchliche Definition berücksichtigt keine leeren Bäume. Ein Baum, der nur aus der Wurzel besteht, hat dann die Höhe 0. Die Höhe gemäß dieser zweiten Version entspricht der Anzahl der *Kanten* des längsten Pfads von der Wurzel zu einem Blatt.

   Die Höhe nach dieser zweiten Version wäre also immer um 1 kleiner als die Höhe entsprechend erster Version.

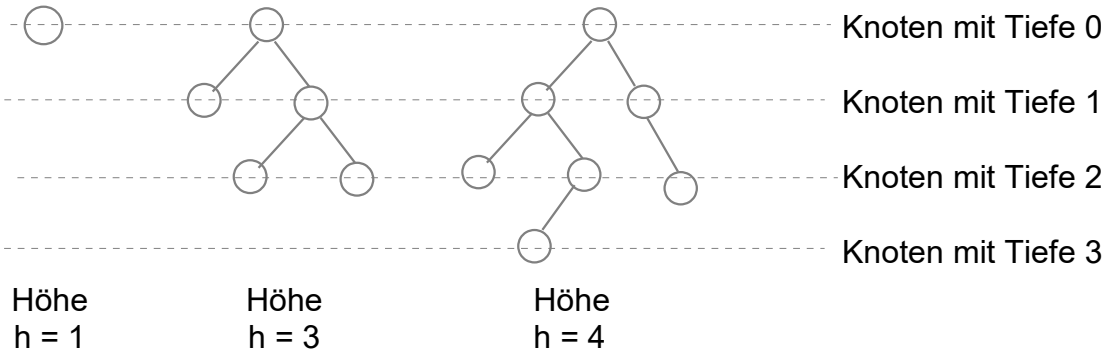
Definition 2.10 - Tiefe eines Knotens im Baums

Die **Tiefe** eines **Knotens** k in einem Baum ist die Länge des Pfads von der Wurzel zum Knoten k

Die Tiefe gibt also an, wie weit der Knoten von der Wurzel entfernt ist. Für unsere Höhendefinition bedeutet das:

- Höhe des Baums = 1 + maximale Tiefe aller Knoten des Baums

Beispiel 2.11 - Höhe und Tiefe



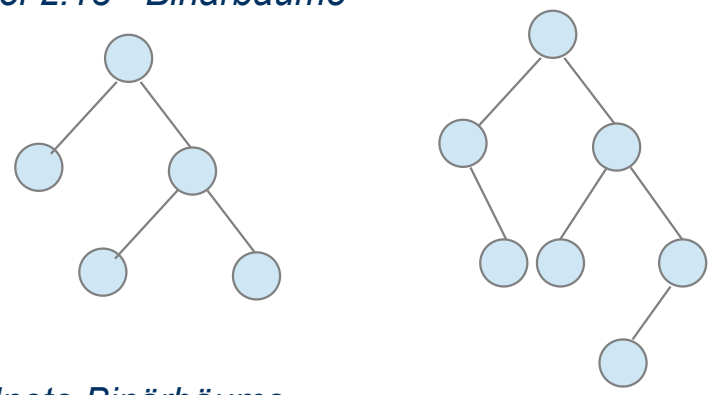
2.4.6 Binärbäume

Oft werden wir in der Informatik eine einfache Art von Bäumen verwenden (z.B. für sog. binäre Suchbäume als Datenstruktur für effiziente Suche), bei denen ein Knoten maximal zwei Kinder haben kann.

Definition 2.12 - Binärbaum

Ein **Binärbaum** ist ein Baum, bei dem jeder Knoten höchstens zwei Kinder hat.

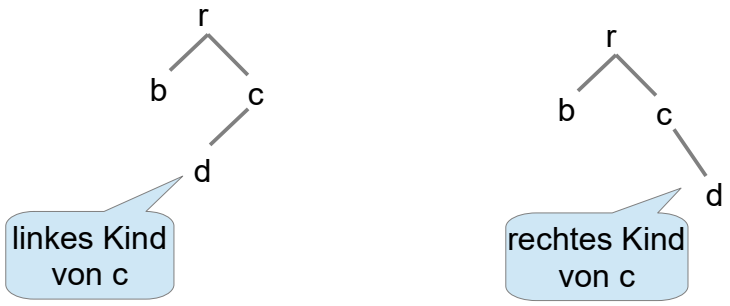
Beispiel 2.13 - Binärbäume



Geordnete Binärbäume

Bei **geordneten Binärbäumen** unterscheidet man bei den Kindern eines Knotens zwischen dem linken und rechten Kind.

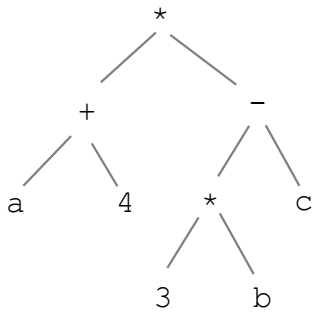
Beispiel: Folgende zwei Bäume sind als geordnete Binärbäume unterschiedlich, als nicht geordnete Binärbaum wären sie aber gleich:



Beispiel 2.14 – Abstrakte Syntaxbäume

Der strukturelle Aufbau von Ausdrücken, die aus Zahlen, Variablen und binären Operatoren bestehen, kann als geordneter Binärbaum dargestellt werden. Diese Darstellung wird als Abstrakter Syntaxbaum (engl. abstract syntax tree, AST) bezeichnet.

Abstrakter Syntaxbaum für Ausdruck `(a+4)*(3*b-c)`:



Aufgabe 2.15 - Abstrakte Syntaxbäume

a) Zeichen Sie einen abstrakten Syntaxbäume für folgenden Ausdruck:

```
4 * (x + 5) - y
```

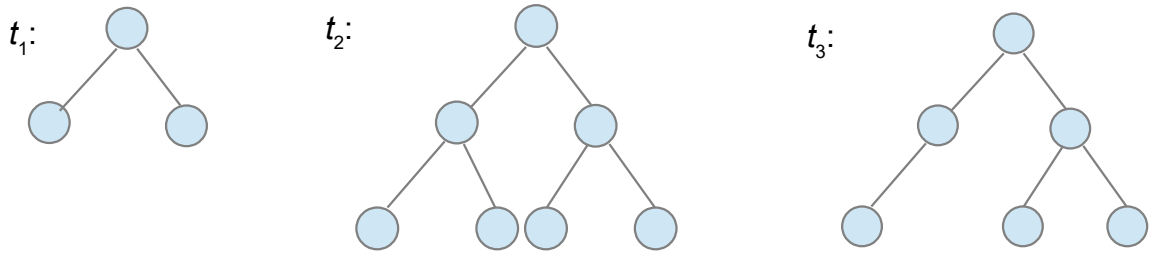
b) Welche Höhe hat der Baum?

c) Welche Tiefe haben die Knoten `x` und `y`?

Definition 2.16 - Vollständige Binärbäume

Ein Binärbaum heißt **vollständig**, wenn gilt:

- Jeder innere Knoten hat genau zwei Kinder.
- Alle Blätter haben die gleiche Tiefe, d.h. die gleiche Entfernung zur Wurzel.



vollständiger Binärbaum der Höhe $h(t_1) = 2$

vollständiger Binärbaum der Höhe $h(t_2) = 3$

kein vollständiger Binärbaum

Für die Analyse der Komplexität von Programmen werden wir folgende Eigenschaften vollständiger Binärbäume benötigen:

Eigenschaft 2.17 - Vollständige Binärbäume

- Ein **vollständiger Binärbaum** der Höhe $h > 0$ hat
  - $2^{(h-1)}$ **Blätter,**
  - $2^h - 1$ **Knoten insgesamt** (innere Knoten und Blätter).
- **Mindesthöhe von Binärbäumen**: Jeder Binärbaum mit ***n* Knoten** ($n > 0$) hat eine Höhe
  $$h \geq \lceil ld(n+1) \rceil$$

- Bei vollständigen Binärbäume verdoppelt sich offensichtlich jedes mal die Anzahl der Blätter, wenn die Höhe um 1 zunimmt. Die Anzahl der Konten insgesamt ergibt sich somit als Summe von Zweierpotenzen (Spezialfall geometrische Reihe, siehe vorne).
- Die Mindesthöhe ergibt sich daraus, dass ein vollständiger Binärbaum bei einer gegebenen Höhe *h* der Binärbaum ist, der die maximale Anzahl an Knoten enthält.

Aufgabe 2.18 - Eigenschaften vollständiger Binärbäume

a) Wie viele *Knoten* enthält ein *vollständiger Binärbaum* der Höhe $h = 10$?

b) Wie viele *Blätter* enthält ein *vollständiger Binärbaum* der Höhe $h = 10$?

c) Ist folgende Aussage wahr oder falsch? "Jeder vollständige Binärbaum enthält mehr Blätter als innere Knoten."

d) Wahr oder falsch? "Es gibt mindestens einen Binärbaum der Höhe 8, der 500 Knoten enthält."

2.4.7 Traversierung von Binärbäumen

Eine typische (rekursive) Verarbeitungsweise bei Binärbäumen ist die, dass alle Knoten eines Binärbaums systematisch durchlaufen werden.

Definition 2.19 - Rekursive Verarbeitung von Binärbäumen

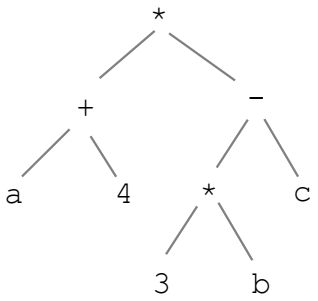
- **Preorder-Traversierung:** falls Baum nicht leer:
  (1) verarbeite Wurzel
  (2) durchlaufe linken Teilbaum in Preorder-Reihenfolge (rekursiv),
  (3) durchlaufe rechten Teilbaum in Preorder-Reihenfolge (rekursiv)
- **Inorder-Traversierung**: falls Baum nicht leer:
  (1) durchlaufe linken Teilbaum in Inorder-Reihenfolge (rekursiv)
  (2) verarbeite Wurzel
  (3) durchlaufe rechten Teilbaum in Inorder-Reihenfolge (rekursiv)
- **Postorder-Traversierung:** falls Baum nicht leer:
  (1) durchlaufe linken Teilbaum in Postorder-Reihenfolge (rekursiv)
  (2) durchlaufe rechten Teilbaum in Postorder-Reihenfolge (rekursiv)
  (3) verarbeite Wurzel

Allen drei Traversierungsfolgen ist gemeinsam, dass bei einem nicht leeren (Teil-)baum, der eine Wurzel w und zwei Teilbäume L und R hat, sowohl die Wurzel als auch (rekursiv) die Knoten in den Teilbäumen bearbeitet werden. Der Unterschied liegt darin, wann die Wurzel drankommt:

Preoder (pre = vor):	**w – L – R**
Inorder (in = dazwischen):	**L – w – R**
Postorder (post = danach):	**L – R – w**

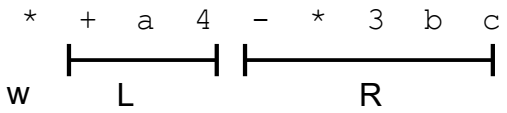
Beispiel 2.20- AST traversieren

Gegeben ist folgender Abstrakter Syntaxbaum für den Ausdruck `(a+4)*(3*b-c)`:



Welche Bearbeitungsreihenfolge ergibt sich bei den verschiedenen Traversierungsreihenfolgen *Preorder*, *Inorder* und *Postorder*?

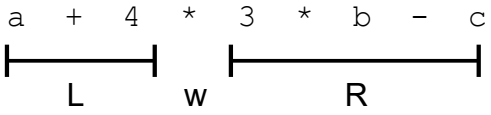
- Preorder:



  Anmerkung: Preorder-Notation ergibt Ergebnis ähnlich funktionale Notation

  `*( +(a,4), -(*(3,b), c))`

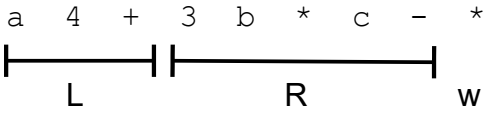
- Inorder:



  Anmerkung: bis auf Klammer ergibt das die übliche Infix-Notation:

  `((a + 4) * ((3 * b) - c))` (mit Klammern um Teilbaum)

- Postorder:



  Anmerkung:

  Postorder ergibt die sog. Umgekehrte Polnische Notation (UPN), die ähnlich im Compilerbau bei der Codegenerierung eine wichtige Rolle spielt. Soll ein Teilausdruck mit einem Operator und zwei Operanden ausgewertet werden, dann müssen erst die Operanden berechnet werden, bevor die Operation ausgeführt werden kann.

Aufgabe 2.21 - Baumtraversierung

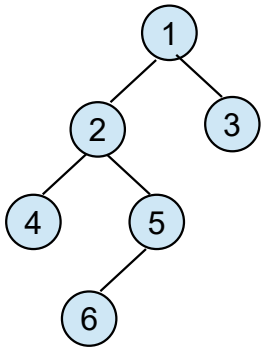
Welche Bearbeitungsreihenfolgen ergeben sich, wenn der folgende Binärbaum per

a) *Preorder*-Durchlauf,

b) *Inorder*-Durchlauf,

c) *Postorder*-Durchlauf

traversiert wird?



Aufgabe 2.22 - Baumtraversierung

Geben Sie jeweils einen Binärbaum der Höhe h=3 an, so dass

a) die **Preorder**-Traversierung

b) die **Inorder**-Traversierung

c) die **Postorder**-Traversierung

die Folge

a, b, c, d, e, f

ergibt.

Fazit zu Lektion 2

Das sollten Sie in dieser Lektion gelernt haben

- Wiederholung: Was ist die arithmetische und die geometrische Reihe?
- Wiederholung: Welche Eigenschaften gelten für Potenzen und Logarithmen?
- Welche Notationskonventionen werden in diesem Semester verwendet?
- Was sind gerichtete oder ungerichtete Graphen und Bäume?
- Was sind Grundbegriffe und Eigenschaften von Graphen und Bäumen?
- Welche Eigenschaften haben vollständige Binärbäume?
- Was versteht man unter Inorder-, Preorder- und Postordertraversierung?